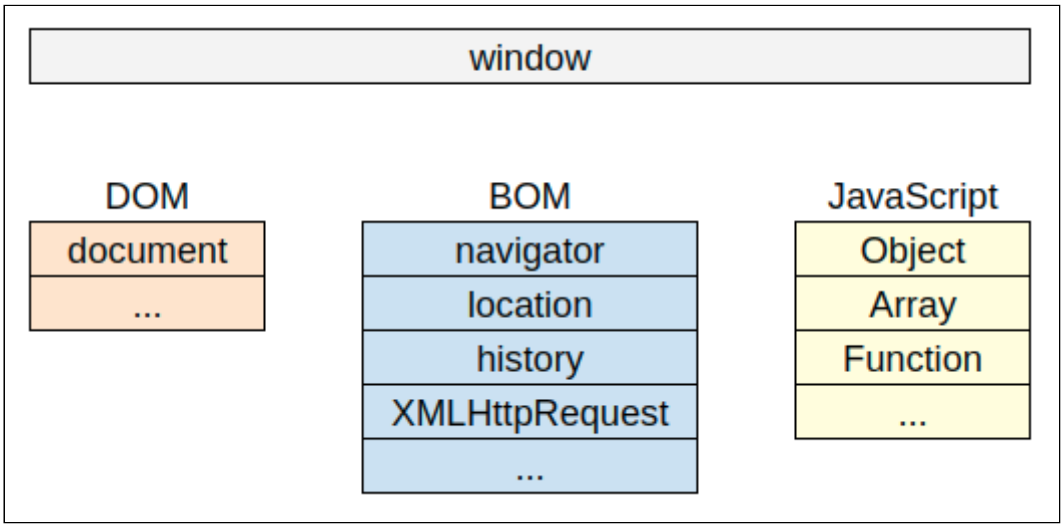
DOM - Document Object Model

- W3Schoolsin DOM-osiota kannattaa silmäillä tämän materiaalin ohella
- DOMin muodostaminen selaimessa - tarkempaa ja pätevämpää selitystä

Selainympäristö



Window-olio

- Selaimen suorittaman web-ohjelmoinnin kannalta window-olio on ylin taso, joka
  - **BOM**in avulla määrittelee selainikkunan ja sen ominaisuudet ja funktiot **selaimen määrittelemänä**
  - **DOM**in avulla mahdollistaa selainikkunaan ladatun dokumentin käsittelyn
  - **JavaScript**-ympäristön käytettäväksi tarjoaa myös viittauksen kaikkiin JavaScriptin globaaleihin muuttujiin, joita ei ole kiinnitetty mihinkään funktioihin

JavaScript ja DOM

- **DOM** (*Document Object Model*) on ohjelmointirajapinta HTML/XML-dokumenttien rakenteen ja sisällön muokkaamiseksi.
- **DOM** eli **dokumenttioliomalli** kuvaa HTML-dokumentin *elementit* ja *selaimen toiminnalliset osat* JavaScriptin kannalta **olioina.**

- Jokainen DOMin osa on solmu (node)
- DOM ei kuulu itse JavaScriptiin
- DOMin avulla dokumentin rakennetta ja sisältöä voidaan läpikäydä useilla erilaisilla metodeilla kuten `firstChild`, `lastChild`, `childNodes` ja `parentNode`.
- Elementtejä ja niiden sisältöjä voidaan noutaa erilaisilla metodeilla kuten esim. `document.getElementById()` tai `document.getElementsByTagName()`.

Document-olio

- Window-olion yhtenä ominaisuutena **document**-olio, joka sisältää koko HTML-dokumentin jäsennettynä puurakenteeksi.

DOM-puu

- Selaimeen ladattu HTML-dokumentti tulkitaan DOM-puuna, joiden solmuja DOMin oliot ovat
- Solmuja muodostava niin HTML-elementit kuin niiden tekstisisällötkin
- Solmuilla on sijaintinsa perusteella esim. `vanhempi-lapsi-` ja `sisaruus-` suhteita, joiden perusteella DOM-puuta on mahdollista käsitellä JavaScriptillä

Esimerkki 0401

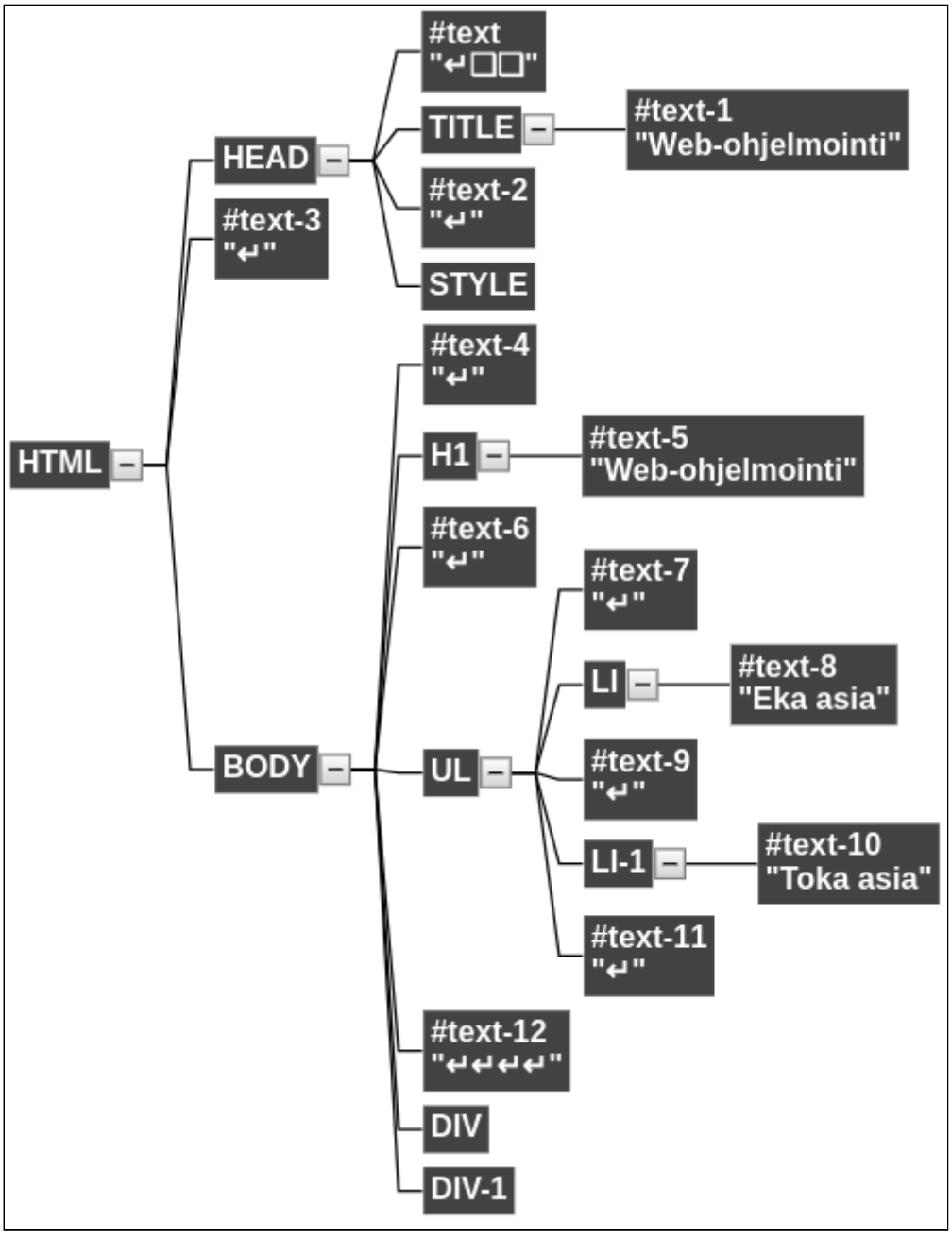
Tarkastellaan seuraavaa HTML-dokumenttia DOM-puuna

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Web-ohjelmointi</title>
</head>
<body>
<h1>Web-ohjelmointi</h1>
<ul>
<li>Eka asia</li>
<li>Toka asia</li>
</ul>

</body>
</html>
```

- Palvelussa http://software.hixie.ch/utilities/js/live-dom-viewer/# voi katsella HTML-dokumenttia solmuista koostuvana DOM-puurakenteena
- Google Chrome -selaimeen voi asentaa pienen *DOM node tree viewer* -lisäosan, jolla voi myös tulkita HTML-dokumenttia DOM-puuna. Havaintoja oheisesta kuvasta

- HTML-solmulla on kolme lasta (child): `HEAD`, `#text-3` ja `BODY`. Huomaa, että tekstisolmulla `#text-3` on sisältönä HEAD-elementin lopputägin jälkeinen rivinvaihtomerkki.
- UL-solmu toimii mm. molempien LI-solmujen vanhempana (parent)
- Solmut `#text-7`, `LI`, `#text-9`, `LI` ja `#text-11` ovat keskenään sisaruksia (siblings)



BOM

- Browser Object Model (BOM) tarjoaa lisäksi sellaisia selaimen tarjoamia olioita, joita käytetään kaikkeen muuhun kuin työskentelemiseen ladatun dokumentin kanssa

- `location`-olion avulla voi esim. lukea nykisen URLin ja ohjata selaimen uuteen osoitteeseen
- `navigator`-olion avulla voi tutkia selaimen versiota jne.
- Tarjolla on myös funktioita mm. ikkunan luomiseen (open) ja sulkemiseen (close).
- Kaikkiin avoinna oleviin ikkunoihin tai kehyksiin voi kirjoittaa mistä tahansa ikkunasta viittaamalla ikkunan nimeen.
- Monenlaista selaimeen liittyvää toiminnallisuutta: `location`, `alert`, `confirm`, `prompt`, `alert`, `setTimeout`...

Elementtien etsiminen ja valinta

- Tarjolla 6 keskeistä metodia `getElementById`, `getElementsByName`, `getElementsByTagName`, `getElementsByClassName`, `querySelector` ja `querySelectorAll`

Esimerkki 0402

- getElementByID-metodilla voidaan valita yksittäinen elementti id-tunnisteen perusteella
- Ohjelma vaihtaa p-elementtiin sisällöksi sinisen tekstin Uusi teksti!

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Alustava poistuva teksti</p>

<script>
let elem = document.getElementById("demo");
elem.innerHTML   = "Uusi teksti!";
elem.style.color = "blue";
</script>

</body>
</html>
```

Esimerkki 0403

- `querySelector`-metodilla voi valita monipuolisesti elementtejä. Huom! Palauttaa vain ensimmäisen löydöksen

```
<!DOCTYPE html>
<html>
<body>

<h2>Otsikko</h2>
```

```
<p>Kappale 1</p>
<p class="sini">Kappale 2</p> <!-- VAIN tälle sininen tausta -->
<p class="sini">Kappale 3</p>

<script>
document.querySelector("p.sini").style.backgroundColor = "blue";
</script>

</body>
</html>
```

Esimerkki 0404

`querySelectorAll`-metodilla voi valita kokoelman elementtejä. Läpikäynti tulee tehdä elementti kerrallaan:

```
<!DOCTYPE html>
<html>
<body>

<h2>Otsikko</h2>
<p>Kappale 1</p>
<p class="sini">Kappale 2</p> <!-- sininen tausta -->
<p class="sini">Kappale 3</p> <!-- sininen tausta -->

<script>
let elem = document.querySelectorAll("p.sini");
for (i = 0; i < elem.length; i++) {
    elem[i].style.backgroundColor = "blue";
}

</script>

</body>
</html>
```

Tämä **ei** toimi, koska **kokoelmalla** ei ole esitettyä style.backgrounColor-ominaisuutta (mutta kokoelman yksittäisellä elementillä on)

```
document.querySelectorAll("p.sini").style.backgroundColor = "blue";
```

Elementtien ja sisällön lisääminen

- Elementin lisääminen: document.createElement
- Tekstisolmun lisääminen: document.createTextNode
- Solmun liittäminen elementtiin viimeiseksi: element.appendChild

Esimerkki 0405

```
//HTML:
<ul id="demolist">
<li>Eka</li>
<li>Toka</li>
</ul>

//JS:
let ulnode = document.getElementById("demolist");
let linode = document.createElement('li');
let litext = document.createTextNode('Koka');
linode.appendChild(litext);
ulnode.appendChild(linode);
```

Tulos:

- Eka
- Toka
- Koka

Esimerkki 0406 - innerHTML

```
//HTML:
<p id="demo"></p>

//JS:
let para = document.getElementById("demo");
para.innerHTML   = "Olen uusi tekstikappaleen sisältö";
```

Huom: Tarjolla on lukuisa määrä erilaisia metodeja solmujen lisäämiseksi haluttuihin paikkoihin DOM-puussa mm. insertBefore, insertBefore, appned, prepend, before, after, ...

Elementtien poistaminen

- Solmun poistaminen: node.remove()
- Lapsielementin poistaminen vanhemmalta: parentElem.removeChild

Esimerkki 0407

- UL-listan lapsisolmujen poistaminen

```
//HTML:
<ul id="demolist">
<li>Eka</li>
<li>Toka</li>
</ul>

//JS:
```

```
var list = document.getElementById("demolist");
while(list.hasChildNodes()) {
    list.removeChild(list.firstChild);
}
```

Tapahtumien käsittely

- Graafisen käyttöliittymän ohjelmointia kutsutaan usein *tapahtumaohjatuksi ohjelmoinniksi* (*event-driven programming*).
- DOM-solmut generoivat erilaisia tapahtumia (events). Yleisimpiä ovat esim.
  - **click** - hiiren klikkaus elementtiin
  - **mouseover**, **mouseout**, **mousedown**, **mouseup**, **mousemove**, **contextmenu** - hiiren generoimia tapahtumia
  - **submit**, **focus** - lomakkeen ja sen kenttien tapahtumia
  - **keydown**, **keyup** - näppäintapahtumia

Tapahtumankäsittelijä

- Elementtiin on asetettava tapahtumankäsittelijä, jos sen halutaan reagoivan tiettyihin tapahtumiin
- Tapahtumankäsittelijä voidaan asettaa kolmella tapaa
  1. HTML-attribuuttina: `onclick="..."`.
  2. DOM-ominaisuutena: `elem.onclick = function`.
  3. Tapahtumankuuntelijametodina: `elem.addEventListener(event, handler[, phase])` Poistaminen: `removeEventListener`

> Huom: Kahdessa ensimmäisessä tavassa tapahtuman eteen kirjoitetaan **on**: click-tapahtumalle **on**click. Ensimmäisessä tavassa attribuutti voidaan kirjoittaa millä tahansa kirjainkoolla onclick, onClick, ONCLICK,
> ... (on HTML-attribuutti) mutta kahdessa jälkimmäisessä tapahtuma on kirjoitettava pienin kirjaimin

Esimerkki 0408

- Tapahtumankäsittelijän asettaminen HTML-attribuuttina

- Kutsumalla omaa funktiota koodia voidaan kirjoittaa enemmän sotkematta HTML-merkkausta

```
// 1. Kaikki toiminta attribuutin arvona
<button onclick="alert('Klikkasit mua')">Klikkaa</button>

// 2. Kutsu omaan funktioon
<p onclick="fn('Klikkasit mua')">Klikkaa</p>

<script>
function fn(teksti) {
    let infoteksti = "Tekstisi oli: " + teksti;
    alert(infoteksti);
}
</script>
```

Esimerkki 0409

- Tapahtumankäsittelijän asettaminen DOM-ominaisuutena

```
<button id="btn1">Klikkaa</button>

<script>
btn1.onclick = function() {
    alert('Klikkasit');
};
</script>
```

Esimerkki 0410

- Tapahtumankäsittelijän asettaminen tapahtumankuuntelijametodina
- Mahdollistaa siistin koodin ja usean kuuntelijan asenttamisen samalla elementille

```
//HTML:
<p id="elem">mouseover/mouseout/click?</a>

//JS:
<script>
function fn1() {
  elem.style.backgroundColor = "yellow";
};

function fn2() {
  elem.style.backgroundColor = "white";
}

function fn3() {
  alert('Klikkasit mua');
};

elem.addEventListener("mouseover", fn1); // tausta keltainen
elem.addEventListener("mouseout", fn2); // tausta valkoinen
```

```
elem.addEventListener("click", fn3); // alert
<script>
```

Esimerkki 0411

- DOM-elementin välittäminen this-viittauksella

```
//HTML:

<p onclick="fn(this)">Klikkaa mua</p>

//JS:
<script>
function fn(elem) {
    elem.innerHTML = "Sähän klikkasit mua!";
}
</script>
```

Tapahtumaolio - EventObject

- Tapahtumasta halutaan usein lisätietoa varsinaisen tapahtumatyypin lisäksi.
- Tapahtumaolio pitää näitä tietoja yllä. Halutaan tietää esim. mitä kohtaa näyttöruudulla hiirellä klikattiin [`event.screenX`, `event.screenY`], mitä näppäintä painettiin [`event.keyCode`] jne.
- Tapahtuman tapahtuessa kutsutaan olion `handleEvent`-metodia automaattisesti. Näin ollen kyseiseen metodiin voi ohjelmoida erilaista toiminnallisuutta riippuen tarkemmista tapahtumatiedosta

Esimerkki 0412

- Kokeile seuraavaa

```
//HTML:
<p id="elem">Klikkaa ja pidä hiiri alhaalla</p>

//JS:
class Valikko {
    handleEvent(event) {
      switch(event.type) {
        case 'mousedown':
          elem.style.backgroundColor = "yellow";
          elem.innerHTML = "Hiiri alhaalla sijainnissa X:"
                         + event.screenX + "Y: " + event.screenY;
          break;
        case 'mouseup':
          elem.style.backgroundColor = "white";
          elem.innerHTML = "Klikkaa ja pidä hiiri alhaalla";
          break;
      }
    }
  }
```

```
let valikko = new Valikko();
elem.addEventListener('mousedown', valikko);
elem.addEventListener('mouseup', valikko);
```

Tapahtumat ja niiden ominaisuudet ja metodit

- Tapahtumat: https://www.w3schools.com/jsref/dom_obj_event.asp
- Tapahtuman ominaisuudet ja metodit: https://www.w3schools.com/jsref/dom_obj_event.asp

Lista tärkeimmistä *tapahtumista*

- blur: Aktiivinen syöttöelementti tulee ei-aktiiviseksi. Siirretään focus esim. seuraavaan syöttökenttään TAB-näppäimellä
- change: Muutetaan jonkin elementin sisältöä.
- click: Klikataan hiiren painikkeella jotain syöttöelementtiä.
- dblclick: Tuplaklikkaus jonkin syöttöelementin päällä.
- focus: Syöte kohdistuu elementtiin. esim. kursorin vienti tekstikenttään aktivoi kentän, minkä jälkeen kaikki näppäimistösyöte siirtyy tekstikenttään.
- keydown: Näppäimistön näppäin painettiin pohjaan. Käytetään parina onkeyup-tapahtuman kanssa.
- keyup: Näppäimistön painikkeen vapautus.
- keypress: Käyttäjä painoi jotain näppäimistön näppäintä (yhdistää onkeydown- ja onkeyup-tapahtumat).
- mousedown: Hiiren painike painetaan alas.
- mouseup: Hiiren painike vapautetaan
- mouseover: Hiiren kursori on jonkin elementin päällä.
- mouseout: Hiiren kursori poistuu jonkin elementin päältä (mouseover-tapahtuman vastinpari).
- reset: Lomakkeen kenttien asetetaan oletuisarvoihinsa (yleensä tyhjätään).
- submit: Lomakkeen data lähetetään action-määritteellä osoitettuun sijaintiin
- load: Dokumentti on kokonaan ladattu.

Lomakkeen ja sen kenttien käyttäminen

- Lomakkeilla ja niiden kentillä on paljon omia ominaisuuksia ja tapahtumia

- JavaScript ei vaadi lomakekenttien käyttämiseksi välttämättämättä ympäröivää FORM-elementtiä
- Lomakkeet ovat viitattavissa esim. **document.getElementById('lomakeid')** tai **document.forms.lomake02** tai **document.forms[1]**
- Lomakkeet sisältämät elementit ovat viitattavissa esim **form.elements.syotekentta** tai lyhyemmin **form.syotekentta** tai **form.elements[2]**
- Lomakekenttien ominaisuuksiin voi viitata normaalisti esim. **input.value**, **select.value**, **textarea.value**, **input.checked** (radio buttonit ja checkboxit), **select.selectedIndex**, ...

Esimerkki 0413

- Kokeile ja tutki. Ks. kommentit

```
//HTML:
<form>
  <input type="checkbox" name="numero" value="12"> 12<br>
  <input type="checkbox" name="numero" value="20"> 20<br>
  <input type="checkbox" name="numero" value="53"> 53</br>
    <button id="btn">Klikkaa</button>
</form>

//JS:
<script>
function fn() {
  let form = document.forms[0]; // Eka ja ainut lomake
  let numerot = form.elements.numero; // Kaikki "numero"-nimetyt elementit
  if (numerot[2].checked) { // Kolmas rastitettu
    alert(numerot[2].value + " checked"); // Kolmannen arvo näytetään
  }
};

btn.addEventListener("click", fn);

</script>
```

Esimerkki 0414

- Kokeile ja tutki. Ks. kommentit

```
//HTML:
<form id="f">
   <input type="text" id="n"><br>
   <button type="button" onclick="tulostaLuku(f.n.value)"> //inputin arvo
     Tulosta luku
   </button>
</form>

<div id="results"></div>

//JS:
<script>
```

```
  function tulostaLuku(a) {
   var div = document.getElementById("results");
   // a on alkuperäisen inputin arvo:
   div.innerHTML = "Luku= " + a + "<br>";
    }

</script>
```

Esimerkki 0415

- Kuten edellinen, mutta parametrina välitetään buttonin isäelementti form
- Kokeile ja tutki. Ks. kommentit

```
//HTML:
<form id="f">

   <input type="text" id="n"><br>

   <button type="button" onclick="tulostaLuku(this.parentElement)">
        Tulosta luku
    </button>

</form>



//JS:
<div id="results"></div>


<script>



  function tulostaLuku(target) {

     var div = document.getElementById("results");

     // target on lomake ja n sen kenttä
     div.innerHTML = "Luku= " + target.n.value + "<br>";

   }

</script>
```

Esimerkki 0416

- Esittelee pudotusvalikon (select) valintojen käyttämistä
- Kokeile ja tutki. Ks. kommentit

```
//HTML:
<select id="valikko">
  <option value="Valitse eurot, Please!">Valitse eurot</option>
  <option value="13">13 euroa</option>
```

```
  <option value="31">31 euroa</option>
</select>

<button onclick="tulosta()">
     Tulosta valittu
</button>

<div id="results"></div>

//JS:
<script>

  function tulosta() {


   var div = document.getElementById("results");
   div.innerHTML = valikko.value; // valitun arvo

   if (valikko.selectedIndex == 1) { // Toinen vaihtoehto "selected"
        alert("Wau uskalsit valita 13eur!");
   }

   valikko.options[0].selected = true; // vaihdetaan ensimmäinen
                                       // vaihtoehto "selected"

}
</script>
```

document.write : Vältä käyttöä

- Huomautetaan vielä `document.write`- ja `document.writeln`-metodien käyttämisen välttämisestä, koska ne ovat ajalta ennen DOMia. Tämä huomautus siksi, koska verkosta löytyy runsaasti lähinnä alkeisesimerkkejä tämän käytöstä.
- document.write kirjoittaa SCRIPT-elementin paikalle nähdyn H1-elementin sisältöineen

```
<script>
document.write("<h1>Haloo Maailma!</h1>");
</script
```

- document.write ei käsittele DOMia, selain vain lisää tekstin DOMiin sivua luotaessa => Ne toimivat vain sivun latautumisen yhteydessä
- Jos document.write-metodia kutsutaan sivun latauksen jälkeen, olemassaoleva dokumentti poistetaan ja luodaan uusi => Ei sovellu DOMin muokkaamiseen. Tämän voit tarkistaa kokeilemalla seuraavaa koodinpätkää

```
<h3>Tämä tuhotaan ja korvataan 2 sec kuluttua...</h3>
<script>
  setTimeout(() => document.write('<h3>...tällä uudella sisällöllä</h3>'), 2000);
</script>
```

Lisätietoja

- Keycodes: http://css-tricks.com/snippets/javascript/javascript-keycodes/
- Testaa keyboard-tapahtumia: http://javascript.info/tutorial/keyboard-events

Jätetty tarkoituksella tyhjäksi.